

华中科技大学

课程实验报告

课程名称: C 语言程序设计实验

专业班级: 计算机类 2110

学 号: U202115652

姓 名: 李嘉鹏

指导教师: 卢萍、毛伏兵

报告日期: 2022 年 1 月 5 日

计算机科学与技术学院

目 录

1 实验 6 指针程序设计实验	1
1.1 程序改错与跟踪调试	1
1.2 程序完善	2
1.3 程序修改替换	5
1.4 程序设计	6
1.5 小结	19
2 实验 7 结构与联合实验	20
2.1 表达式求值的程序验证	20
2.2 源程序修改替换	21
2.3 程序设计	23
2.4 小结	42
参考文献	43

1 实验 6 指针程序设计实验

1.1 程序改错与跟踪调试

```
#include <stdio.h>

char *strcpy(char*, const char*);

int main(void) {
    char *s1, *s2, *s3;
    scanf("%s", s2);
    strcpy(&s1, &s2);
    printf("%s\n", s1); //把 s2 给 s1
    scanf("%s", s2);
    strcpy(&s3, &s2);
    printf("%s\n", s3); //把 s2 给 s3
    return 0;
}

char *strcpy(char*t, const char*s) {
    while(*t++=*s++);
    return (t);
}
```

改错前源代码如上所示。在 Dev-C++ 中进行调试并观察程序执行过程中字符数组 s1、s2、s3 的值，结果如下：（假设输入的 s2 为 “abc”）

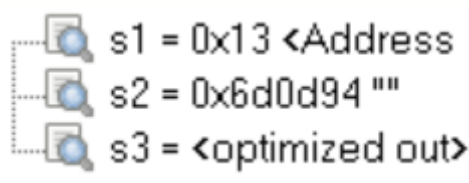


图 1-1 程序跟踪调试题 1 的变量观察图

可见 s1、s2 都存储的是某个地址，在用 strcpy 函数复制的过程中不能有效的复制字符串本身。因此 strcpy 中的形参正确表示方式应该类似于（&s1，&s2），从而间接访问其指向的字符串。

1.2 程序完善

1. 升序排序字符串

```
#include<stdio.h>

#include<malloc.h>

#include<string.h>

void strsort(char *s[], int size){

    char * temp;

    int i, j;

    for(i=0; i<size-1; i++)

        for(j=0; j<size-i-1; j++)

            if(strcmp(s[j], s[j+1])>0) {

                temp=s[j];

                s[j]=s[j+1];

                s[j+1]=temp;

            }

}

int main() {

    int i, N;

    scanf("%d\n", &N);

    char *s[N], t[50];

    for(i=0; i<N; i++) {

        gets(t);

        s[i]=(char *)malloc(strlen(t)+1);

        strcpy(s[i], t);

    }

    strsort(s, N);

    for(i=0; i<N; i++)

        puts(s[i]);

    return 0;

}
```

```
}
```

运行结果如下：



2. 菜单调用实现字符串操作

```
#include<stdio.h>
#include<string.h>
int main(void) {
    char *(*p) (char*, const char *);//函数指针，返回值为
```

指针；p 是指向这样一个函数的指针

```
char a[80],b[80],*result;
int choice;
while(1) {
    do{
        scanf("%d",&choice);
    }while(choice<1||choice>4);
    switch(choice) {
        case 1: p=strcpy; break;
```

```

        case 2:            p=strcat; break;
        case 3:            p= strtok; break;
        case 4:            goto down;

    }

    getchar();
    gets(a);
    gets(b);
    result=(*p)(a,b);
    printf("%s\n",result);
}

down:
return 0;
}

```

运行结果如下：

▼ 测试集1
消耗内存23.3MB
代码执行时长: 0.01秒


测试输入:
1
the more you learn,
the more you get.
2
the more you learn,
the more you get.
3
www.educoder.net
.
4

—— 预期输出 ——

the more you get.
the more you learn,the more you get.
www

—— 实际输出 ——

the more you get.
the more you learn,the more you get.
www

[展示原始输出](#)

▼ 测试集3 消耗内存23.3MB 代码执行时长: 0.01秒

测试输入: 1
hello
world
2
hello
world
3
aaabcccbddd
b
4

—— 预期输出 —— 实际输出 —— 展示原始输出

world
helloworld
aaa world
helloworld
aaa

1.3 程序修改替换

1. 升序排序字符串

由于对于一个指针本身可以进行加减运算，从而实现所指对象的改变（相当于前移或后移），因此可以使用二级指针形参重写 strcmp 函数，本质上还是数组元素的替换。替换后的函数部分代码如下：

```
if(strcmp(*(s+j),*(s+j+1))>0){  
    temp=*(s+j);  
    *(s+j)=*(s+j+1);  
    *(s+j+1)=temp;//对两个地址进行交换  
}
```

运行结果如下：

▼ 测试集1 消耗内存23.3MB 代码执行时长: 0.01秒

测试输入: 3
C
Python
Java

—— 预期输出 —— 实际输出 —— 展示原始输出

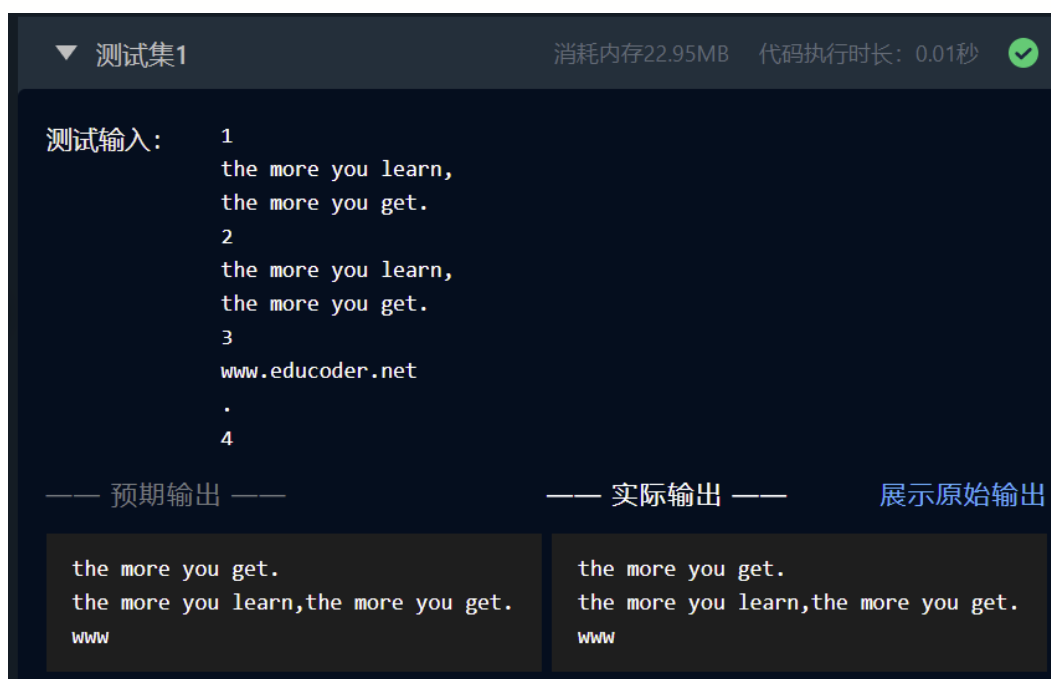
C
Java
Python C
Java
Python

2. 菜单调用实现字符串操作

题目要求使用转移表代替 switch 函数的功能，可以定义一个函数指针数组（其中 pf 是函数指针，它有三个指向有两个形参分别为 char* 和 const char*、返回值为指针的指针函数的指针元素，分别代表三种不同的功能）。修改部分代码如下：

```
char*(*pf[3])(char*, const char*)={strcpy, strcat, strtok};  
.....  
result=(*pf[choice-1])(a,b);
```

运行结果如下：



The screenshot shows a code execution interface with a dark theme. At the top, it says '测试集1' (Test Set 1) with a dropdown arrow, '消耗内存22.95MB' (Consumed memory 22.95MB), '代码执行时长: 0.01秒' (Code execution time: 0.01s), and a green checkmark icon. Below this, the '测试输入:' (Test Input) section shows four lines of input: '1', 'the more you learn, the more you get.', '2', 'the more you learn, the more you get.', '3', 'www.educoder.net', and '4'. Below the input, there are three tabs: '预期输出' (Expected Output), '实际输出' (Actual Output), and '展示原始输出' (Show Original Output). The '预期输出' and '实际输出' tabs are active, showing the same text: 'the more you get.', 'the more you learn,the more you get.', and 'www'.

1.4 程序设计

1. 利用指针取出字节

思路：因为要从高字节开始依次取出每字节的高、低 4 位并显示，所以可以先利用一个指针指向要操作的整形变量，然后通过循环改变指针所指对象，进行交集运算并依次输出。代码如下：

```
#include<stdio.h>  
  
#include<string.h>  
  
int main() {  
    int n;
```



```

int k;
scanf("%d",&n);
char *p=(char*) &n;//给 p 赋初始地址
for(k=3;k+1;k--) {
    printf("%x ",((unsigned char) (*(p+k)))>>4);
    printf("%x", (*(p+k))& 0x0f);
    if(k!=0)
        printf(" ");
}
}

```

运行结果如下：

▼ 测试集1 消耗内存24.43MB 代码执行时长: 0.01秒 ✓

测试输入: 123456

—— 预期输出 —— 实际输出 —— 展示原始输出

0 0 0 1 e 2 4 0 0 0 0 1 e 2 4 0

2. 删除重复元素

思路：采用“两个指针”的方式（一个读下一位是否有重复元素，一个写下一位的元素），定义一个函数 `RemoveSame(a, n)` 去掉重复元素并引入计数器 `count`，返回去重后序列的长度。代码如下：

```

#include<stdio.h>

int RemoveSame(int *a, int n)//去掉重复元素
{
    int i, j;
    int count=0;//重复元素个数
    for(i=0, j=1; j<n; j++) {
        if(a[i]!=a[j]) {
            a[++i]=a[j];
        }
    }
}

```

```

        count++;
    }

}

return count+1;//去重之后的数组元数个数是 count 加上 1(第一个数)
}

int main() {
    int num, i;
    scanf("%d", &num);
    int s[50];
    for(i=0; i<num; i++) {
        scanf("%d", &s[i]);
    }

    num=RemoveSame(s, num);
    for(i=0; i<num-1; i++) {
        printf("%d ", s[i]);
    }

    printf("%d", s[num-1]);
    printf("\n%d", num);
    return 0;
}

```

运行结果如下：

▼ 测试集1
消耗内存24.63MB
代码执行时长：0.01秒


测试输入：

5

3 3 3 6 6

—— 预期输出 ——

3 6

2

—— 实际输出 ——

3 6

2

[展示原始输出](#)



3. 旋转图像

思路：此处“图像”的旋转本质上是矩阵逆时针旋转 90° ，因此定义一个二维数组，读入数组元素后按照旋转后的规律输出即可。代码如下：

```
#include<stdio.h>

int main() {
    int n, m, i, j;
    scanf("%d %d", &n, &m);
    char a[n][m];
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++)
            scanf("%d", &a[i][j]);
    }
    for(i=m-1; i>=0; i--) {
        for(j=0; j<n; j++) {
            if(j!=(n-1))
                printf("%d ", a[j][i]);
            else
                printf("%d\n", a[j][i]);
        }
    }
    return 0;
}
```

运行结果如下：



4. 命令行实现对整数排序

思路：由于需要通过命令行输入参数实现功能，所以 main 函数的形参是 int argc 和 char* argv[]。然后利用 atoi 读取命令行输入的第二个参数（即需要排序的整数个数），读入这些整数后进行选择排序，再依据命令行是否输入第三个参数-d 来决定升序或降序输出。同时，为了使整数的存储没有冗余，可以用 malloc 开辟内存空间，每输入一个整数就开辟一次。代码如下：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main(int argc, char* argv[]) {
    int i, j, temp;
    int n=atoi(argv[1]);
    int *num=(int *)malloc(n*sizeof(int));
    for(i=0;i<n;i++) {
        scanf("%d",&num[i]);
    }
    for(i=0;i<n-1;i++)//进行 n-2 轮排序
    {
        for(j=i+1;j<n;j++) {
```

```

        if(num[i]<num[j]){
            temp=num[i];
            num[i]=num[j];
            num[j]=temp;
        }
    }
}

if(strcmp(argv[argc-1],"-d")==0){
    for(i=0;i<n;i++){
        printf("%d  ",num[i]);
    }
}
else{
    for(i=n-1;i>=0;i--){
        printf("%d  ",num[i]);
    }
}

return 0;
}

```

运行结果如下：

▼ 测试集1
消耗内存22.83MB
代码执行时长：0.07秒

测试输入： ./sort 5 -d
4 3 8 5 1

预期输出

实际输出

展示格式化输出

8 5 4 3 1

8 5 4 3 1

▼ 测试集2
消耗内存22.83MB
代码执行时长：0.08秒

测试输入： ./sort 12
4 3 8 5 1 3 6 7 8 9 10 12

预期输出

实际输出

展示格式化输出

1 3 3 4 5 6 7 8 8 9 10 12

1 3 3 4 5 6 7 8 8 9 10 12

5. 删除子串

思路：要删除字符串中出现的所有子串，可以设计两个函数：一个 strStr 用来遍历搜索字符串中出现的每一个子串，一个 delSubstr 用来删除子串，最后在 main 函数中调用即可。

整个过程的流程图如下：

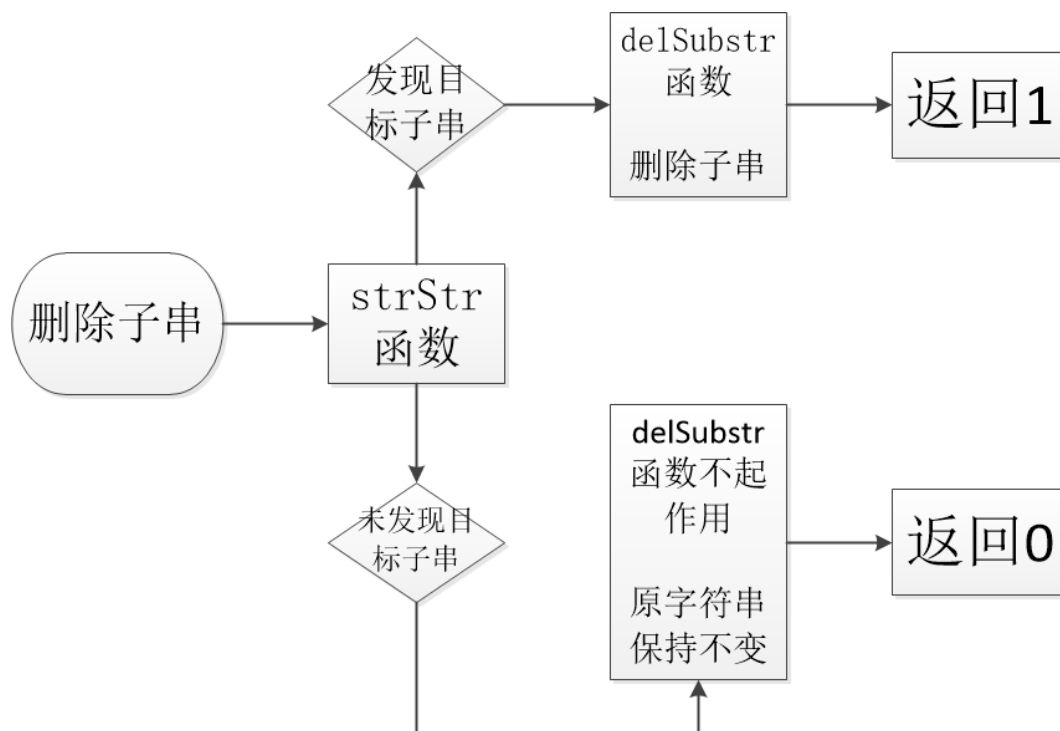


图 1-2 程序设计题 5 的流程图

代码如下：

```
#include <stdio.h>

#include <string.h>

void strStr(char *a, char *t, int *res){

    int m;

    int k;

    for(m=0;*(a+m)!='\0';m++){

        if(*(m+a)==*t){

            k=1;

            while(*(a+k+m)==*(k+t)&&*(t+k)!='\0'){

                k+=1;

            }

            *res=m;

            return;

        }

    }

    *res=-1;

}
```

```

        }
        if(k==strlen(t))
            *(res+m)=1;
    }
}

int delSubstr(char *str, char *substr) {
    int res[80]={};
    int i,j;
    int *p=res;
    strStr(str, substr, res);
    for(i=0;i<80;i++) {
        if(*(p+i)==1)
            for(j=i;j<i+strlen(substr);j++)
                str[j]=' \0' ;
    }
}

int main() {
    int q;
    char str[80]={};
    char substr[80]={};
    char *pS=str;
    gets(str);
    gets(substr);
    int len=strlen(str);
    delSubstr(str, substr);
    for(q=0;q<80;q++)
        putchar(*(pS+q));
    if(strlen(str)!=len)
        printf("\n1");
}

```

```

else
    printf("\n0");
return 0;
}

```

运行结果如下：

▼ 测试集1

消耗内存25.35MB 代码执行时长：0.01秒

测试输入：

stay hungry stay foolish

stay

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

hungry foolish

1

hungry foolish

1

▼ 测试集2

消耗内存25.35MB 代码执行时长：0.01秒

测试输入：

Talk is cheap,show me the code.

abcd

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

Talk is cheap,show me the code.

0

Talk is cheap,show me the code.

0

▼ 测试集7

消耗内存25.35MB 代码执行时长：0.01秒

测试输入：

aabcbccabc

abc

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

abcc

1

abcc

1

6. 求超长非负整数积

思路：由于不可能直接表示出两个极大数的乘积，可以考虑用字符型数组来存储所要计算的两个大数（char a,b），然后采用手动计算的方法。两个数先对齐，再按位相乘，分别讨论进位和不进位的情况进行乘法运算。代码如下：

```

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

```

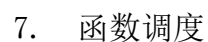


```

void mul(char *a, char *b, char *c){
    int i, j, ca, cb, *s;
    ca=strlen(a);
    cb=strlen(b);
    if(strcmp(a, "0")==0 || strcmp(b, "0")==0)//如果两个数中含有 0
    {
        strcpy(c, "0");
        return;
    }
    s=(int *)malloc(sizeof(int)*(ca+cb));
    for(i=0; i<ca+cb; i++)
        s[i]=0;//初值
    for(i=0; i<ca; i++)
        for(j=0; j<cb; j++)
            s[i+j+1]+=(a[i]-'0')*(b[j]-'0');//不考虑进位做乘法
    for(i=ca+cb-1; i>=0; i--)//进位
        if(s[i]>=10){
            s[i-1]+=s[i]/10;
            s[i]%=10;
        }
    i=0;
    while(s[i]==0)//如果结果前面都是 0
        i++;
    for(j=0; i<ca+cb; j++, i++)//数值转化为字符并存储
    {
        c[j]=s[i]+'0';
    }
    c[j]='\0';//字符串识别
    free(s);
}

```

运行结果如下：



1

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
typedef void(*pt)(); //定义 pt 为函数指针
void task0() {
    printf("task0 is called!\n");
}
void task1() {
    printf("task1 is called!\n");
}
void task2() {
    printf("task2 is called!\n");
}
void task3() {
    printf("task3 is called!\n");
}
void task4() {
    printf("task4 is called!\n");
}
void task5() {
    printf("task5 is called!\n");
}
void task6() {
    printf("task6 is called!\n");
}
void task7() {
    printf("task7 is called!\n");
}

```

```
pt fun[9]={task0, task1, task2, task3, task4, task5, task6, task7}; //函数指针数组
```

```
void scheduler ()//调度函数
```

```
{
    pt fun[100];
    int len;
    char s[1000];
    scanf("%s", s);
    len=strlen(s);
    for(int i=0;i<len;i++){
        int temp;
        temp=s[i]-'0';
        if(temp==0) fun[i]=task0;
        else if(temp==1) fun[i]=task1;
        else if(temp==2) fun[i]=task2;
        else if(temp==3) fun[i]=task3;
        else if(temp==4) fun[i]=task4;
        else if(temp==5) fun[i]=task5;
        else if(temp==6) fun[i]=task6;
        else if(temp==7) fun[i]=task7;
    }
```

```
execute(fun, len);
```

```
}
```

```
void execute(pt* fun, int len)//函数执行
```

```
{
    for(int i=0;i<len;i++){
        pt pfun=fun[i];
        pfun();
    }
```

```

}

int main() {
    scheduler();
    return 0;
}

```

运行结果如下：

测试集1
消耗内存23.07MB
代码执行时长：0.01秒

测试输入： 13607122

预期输出

task1 is called!
task3 is called!
task6 is called!
task0 is called!
task7 is called!
task1 is called!
task2 is called!
task2 is called!

实际输出

task1 is called!
task3 is called!
task6 is called!
task0 is called!
task7 is called!
task1 is called!
task2 is called!
task2 is called!

[展示原始输出](#)

测试集2
消耗内存23.07MB
代码执行时长：0.01秒

测试输入： 1350

预期输出

task1 is called!
task3 is called!
task5 is called!
task0 is called!

实际输出

task1 is called!
task3 is called!
task5 is called!
task0 is called!

[展示原始输出](#)

1.4 小结

通过实验，我进一步加深了对指针、函数指针、指针函数、指针的声明、赋值、调用等知识的了解，明白了各种指针直接的区别与联系；同时，在整个过程中由于粗心大意曾经遇到过不少问题。可见，指针是一把“双刃剑”，如果利用得好，可以大大提高程序的效率，但必须注意相关细节，否则可能引发连锁反应。

2 实验 7 结构与联合实验

2.1 表达式求值的程序验证

对于以下六个表达式：（表达式之间相互无关）

(1) $(++p) \rightarrow x$, 是括号内优先, p 先自增, 然后指向 $a[1]$, 对应的成员 x 为 100.

(2) $p++$, $p \rightarrow c$, p 先自增, 然后指向 $a[1]$, 对应的成员 c 为 'B'.

(3) $*p++ \rightarrow t$, $*p \rightarrow t$, p 先指向 t , 然后 p 自增, 再指向 $a[1]$ 中的成员 t ("xyz") 的首地址, 结果为 'x'.

(4) $* (++p) \rightarrow t$, p 先自增, 然后指向 $a[1]$ 中的成员 t 的首地址, 结果为 'x'.

(5) $* ++p \rightarrow t$, p 先指向 t , 然后 p 自增, 相当于 $(* (++ (p \rightarrow t)))$, 结果为 "V".

(6) $++ *p \rightarrow t$, p 先指向 $a[0]$ 中的成员 t ("UVWXYZ") 的首地址 ('U'), 然后后移一位, 相当于 $++ (* (p \rightarrow t))$, 结果为 "V".

验证程序如下:

```
#include<stdio.h>

struct T{
    int x;
    char c;
    char *t;
}a[]={ {11, 'A', "UVWXYZ"}, {100, 'B', "xyz"} }, *p=a;

int main() {
    int n;
    scanf("%d", &n);
    int ans1;
    char ans2;
```

```

switch(n) {
    case 1: ans1=(++p)->x; printf("%d",ans1); break;
    case 2: *p=a; p++; p->c; ans2=p; printf("%s",ans2); br
eak;
    case 3: *p=a; *p++->t; *p->t; ans2=p; printf("%s",ans2);
break;
    case 4: *p=a; * (++p) ->t; ans2=p; printf("%s",ans2); break;
    case 5: *p=a; *++p->t; ans2=p; printf("%s",ans2); break;
    case 6: *p=a; ++*p->t; ans2=p; printf("%s",ans2); break;
    default: break;
}

return 0;
}

```

运行结果如下：



2.2 源程序修改替换

1. 替换方案：create_list 的形参 struct s_list *headp 应该改为 struct s_list **headp。同时，要使头指针 headp 指向新创建的链表。修改后的部分代码如下：

```
void create_list(struct s_list **headp, int *p);
```

```

..... (中间不变)
*headp=loc_head;
}

```

运行结果如下：

▼ 测试集1
消耗内存22.81MB
代码执行时长：0.01秒
✓

测试输入：1 2 3 4 5

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

1 2 3 4 5

1 2 3 4 5

▼ 测试集2
消耗内存22.81MB
代码执行时长：0.01秒
✓

测试输入：12 34 66 88 99 103 122 0

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

12 34 66 88 99

12 34 66 88 99

2. 替换方案：如要创建一个后进先出的链表，只需要改变 create_list 函数中的循环部分，让头指针始终指向最后创建的结点，后建结点指向先建结点，先建结点始终是尾结点即可。修改后的部分代码如下：

```

void create_list(struct s_list **headp,int *p){
..... (中间不变)
while (*p){
    loc_head=(struct s_list *)malloc(sizeof(struct s_list));
    loc_head->data=*p++;
    loc_head->next=tail;
    tail=loc_head;
}

*headp=loc_head;
}

```


运行结果如下：

▼ 测试集1

消耗内存25.17MB 代码执行时长：0.01秒

测试输入： 1 2 3 4 5 6 7 8 0

—— 预期输出 ——

—— 实际输出 ——

[展示原始输出](#)

8 7 6 5 4

8 7 6 5 4

▼ 测试集2

消耗内存25.17MB 代码执行时长：0.01秒

测试输入： 122 103 99 88 66 34 12 0

—— 预期输出 ——

—— 实际输出 ——

[展示原始输出](#)

12 34 66 88 99

12 34 66 88 99

2.3 程序设计

1. 设计字段结构

思路：首先声明字段结构，将一个 8 位无符号字节声明为 8 个字段，分别为 bit0-bit7；然后定义函数指针数组，将对应的 8 个函数存在其中，需要时分别调用其指向的函数 f0-f7 即可。代码如下：

```
#include<stdio.h>

struct bits{
    unsigned char bit0:1;
    unsigned char bit1:1;
    unsigned char bit2:1;
    unsigned char bit3:1;
    unsigned char bit4:1;
    unsigned char bit5:1;
    unsigned char bit6:1;
    unsigned char bit7:1;
```

```

}t;

union tttt{

    struct bits a;

    unsigned char b;

};

void f0(){

    printf("the function 0 is called!\n");

}

void f1(){

    printf("the function 1 is called!\n");

}

void f2(){

    printf("the function 2 is called!\n");

}

void f3(){

    printf("the function 3 is called!\n");

}

void f4(){

    printf("the function 4 is called!\n");

}

void f5(){

    printf("the function 5 is called!\n");

}

void f6(){

    printf("the function 6 is called!\n");

}

void f7(){

    printf("the function 7 is called!\n");

}

```


```

int main(void) {
    union tttt y;
    scanf("%d",&y.b);
    void(*p_fun[8])(void)={f0,f1,f2,f3,f4,f5,f6,f7};
    if(y.a.bit0) p_fun[0]();
    if(y.a.bit1) p_fun[1]();
    if(y.a.bit2) p_fun[2]();
    if(y.a.bit3) p_fun[3]();
    if(y.a.bit4) p_fun[4]();
    if(y.a.bit5) p_fun[5]();
    if(y.a.bit6) p_fun[6]();
    if(y.a.bit7) p_fun[7]();
    return 0;
}

```

运行结果如下：

▼ 测试集1

消耗内存23.63MB 代码执行时长：0.01秒 

测试输入： 123

—— 预期输出 ——

the function 0 is called!
the function 1 is called!
the function 3 is called!
the function 4 is called!
the function 5 is called!
the function 6 is called!

—— 实际输出 —— [展示原始输出](#)

the function 0 is called!
the function 1 is called!
the function 3 is called!
the function 4 is called!
the function 5 is called!
the function 6 is called!

▼ 测试集3

消耗内存23.63MB

代码执行时长: 0.01秒

测试输入: 255

—— 预期输出 ——

—— 实际输出 ——

[展示原始输出](#)

the function 0 is called!
the function 1 is called!
the function 2 is called!
the function 3 is called!
the function 4 is called!
the function 5 is called!
the function 6 is called!
the function 7 is called!

the function 0 is called!
the function 1 is called!
the function 2 is called!
the function 3 is called!
the function 4 is called!
the function 5 is called!
the function 6 is called!
the function 7 is called!

2. 班级成绩单

思路与流程图如下图所示：

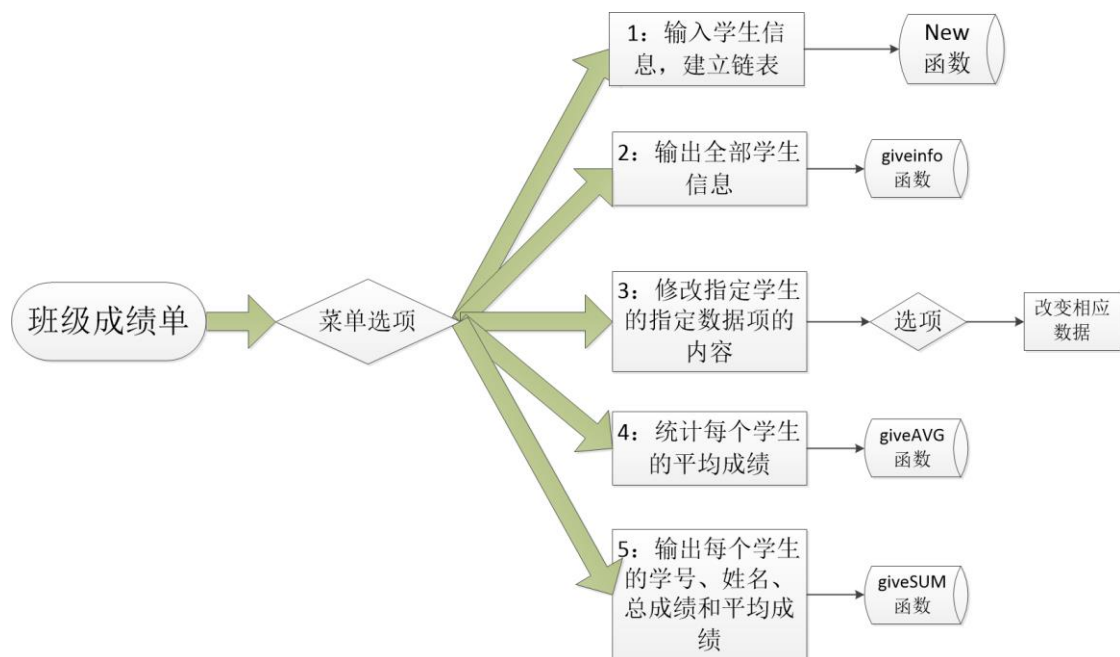


图 2-1 程序设计题 2 的流程图

代码如下：

```

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

```

```

#define POINT_SUM (p->s1 + p->s2 + p->s3 + p->s4)
#define POINT_AVG (POINT_SUM/4.0)
struct stuinfo//定义一个学生信息的结构体
{
    char id[20];
    char name[100];
    float s1, s2, s3, s4;//记录每一门的分数
    struct stuinfo *next;//指向下一个节点的指针
};

void new(struct stuinfo **head, struct stuinfo **tail, int i);
void giveinfo(struct stuinfo *head);
void change(struct stuinfo *head, char *str);
void giveAVG(struct stuinfo *head);
void giveSUM(struct stuinfo *head);

void main() {
    char str[20]={};
    int menu;
    int t=1;
    struct stuinfo *head=NULL, *tail=NULL;
    head=(struct stuinfo*)malloc(sizeof(struct stuinfo));
    while((scanf("%d", &menu))!=EOF) {
        switch(menu) {
            case 1:{
                scanf("%d", &menu);
                if(t!=0) {
                    new(&head, &tail, menu);
                    t=0;
                }
            }
        }
    }
}

```

```

        else
            new(&tail, &tail, menu);
        break;
    }
    case 2:
        giveinfo(head);
        break;
    case 3:
        scanf("%s", str);
        change(head, str);
        break;
    case 4:
        giveAVG(head);
        break;
    case 5:
        giveSUM(head);
        break;
    }
}

return;
}

```

```

void new(struct stuinfo **head, struct stuinfo **tail, int i)
{
    struct stuinfo *p;
    int count=0;
    p=*head, *tail=p;
    do{
        p=(struct stuinfo*)malloc(sizeof(struct stuinfo));

```

```

        scanf("%s %s %f %f %f %f", p->id, p->name, &p->s1
,   &p->s2, &p->s3, &p->s4);
        (*tail)->next=p;
        *tail=p;
        count++;
    }while(count<i);
    (*tail)->next=NULL;
}

```

```

void giveinfo(struct stuinfo *head)
{
    struct stuinfo* p=head->next;
    while(p!=NULL)
    {
        printf("%s %s %.f %.f %.f %.f\n",p->id, p->name
,   p->s1, p->s2, p->s3, p->s4);
        p=p->next;
    }
}

```

```

void change(struct stuinfo *head, char *str)
{
    int menu;
    struct stuinfo *p=head->next;
    while(p!=NULL) {
        if(!(strcmp(p->id, str)))
            break;
        p=p->next;
    }
}

```

```

scanf("%d", &menu);
switch(menu) {
    case 1:
        scanf("%f", &p->s1);
        break;
    case 2:
        scanf("%f", &p->s2);
        break;
    case 3:
        scanf("%f", &p->s3);
        break;
    case 4:
        scanf("%f", &p->s4);
        break;
}
}

void giveAVG(struct stuinfo *head)
{
    struct stuinfo *p=head->next;
    while(p!=NULL) {
        printf("%s %s %.2f\n", p->id, p->name, POINT_AVG);
        p=p->next;
    }
}

void giveSUM(struct stuinfo *head)
{
    struct stuinfo *p=head->next;

```



```

while(p!=NULL) {
    printf("%s  %s  %.0f  %.2f\n",p->id,  p->name,  POINT
_SUM,  POINT_AVG);
    p=p->next;
}
}

```

运行结果如下：

▼ 测试集1

消耗内存20.89MB 代码执行时长：0.01秒

测试输入：

1
2
U202012345 Jack 99 100 80 96
U202054321 Rose 89 94 85 100
2
3
U202054321 1 66
4
5
0

—— 预期输出 ——

U202012345 Jack 99 100 80 96
U202054321 Rose 89 94 85 100
U202012345 Jack 93.75
U202054321 Rose 86.25
U202012345 Jack 375 93.75
U202054321 Rose 345 86.25

—— 实际输出 ——

U202012345 Jack 99 100 80 96
U202054321 Rose 89 94 85 100
U202012345 Jack 93.75
U202054321 Rose 86.25
U202012345 Jack 375 93.75
U202054321 Rose 345 86.25

展示原始输出

▼ 测试集3

消耗内存20.89MB 代码执行时长: 0.01秒

测试输入:

1
1
U123456789 Elio 34 56 78 90
2
3
U123456789 2 99
1
3
U987654321 Gray 0 0 0 0
U000000000 Black 100 100 100 100
U202073456 Red 45 34 67 99
5
0

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

U123456789 Elio 34 56 78 90
U123456789 Elio 301 75.25
U987654321 Gray 0 0.00
U000000000 Black 400 100.00
U202073456 Red 245 61.25

U123456789 Elio 34 56 78 90
U123456789 Elio 301 75.25
U987654321 Gray 0 0.00
U000000000 Black 400 100.00
U202073456 Red 245 61.25

3. 班级成绩单（成绩排序1）

思路：要增添一个为成绩排序的功能，只需要增加一个 sort 函数，通过冒泡排序法进行排序，并交换结点数据域。代码如下：

```
void *sort(struct stuinfo **head)
{
    struct stuinfo *pa, *pb, *pc;
    int flag=0;
    if((*head)->next)
    for(pa=*head; pa->next->next; pa=pa->next){
        pb=pa->next;
        pc=pb->next;
        if(pb->POINT_GIVEAVG < pa->POINT_GIVEAVG){
            flag+=1;

```

```

        pa->next=pb->next;
        pb->next=pa;
        *head=pb;
    }

    if(pc->POINT_GIVEAVG < pb->POINT_GIVEAVG) {
        flag+=1;
        pa->next=pc;
        pb->next=pc->next;
        pc->next=pb;
    }

}

if(flag!=0)
sort(head);
}

```

运行结果如下：

测试集1

消耗内存22.95MB 代码执行时长：0.01秒

测试输入：

1
2
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
2
3
U202054321 1 66
4
5
0

—— 预期输出 ——

U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75

—— 实际输出 ——

U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75

展示原始输出

测试集2

消耗内存22.95MB

代码执行时长: 0.01秒

测试输入:

1
4
U202054321 Rose 89 94 85 100
U202056789 Tom 12 34 56 78
U202012345 Jack 99 100 80 96
U202098765 Jerry 98 76 54 32
2
3
U202054321 1 66
4
5
0

—— 预期输出 ——

U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202056789 Tom 45.00
U202098765 Jerry 65.00
U202054321 Rose 86.25
U202012345 Jack 93.75
U202056789 Tom 180 45.00

—— 实际输出 ——

U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202056789 Tom 45.00
U202098765 Jerry 65.00
U202054321 Rose 86.25
U202012345 Jack 93.75
U202056789 Tom 180 45.00

展示原始输出

4. 回文字符串

思路：要用一个单链表存储字符串，就可以设计一个创建先进先出链表的函数。然后再遍历该链表中位置对称的元素（字母），从而判定是否为回文字符串。

代码如下：

```
void createLinkList(C_NODE **headp, char s[]){
/***** BEGIN *****/第一段代码/
    struct c_node *p, *tail;
    p=(struct c_node *)malloc(sizeof(struct c_node)); //分配空间
    *headp=p;
    tail=p;
    p->data=*s++;
    while(*s!='\0' &&*s!=EOF){
```

```

        tail->next=(struct  c_node  *)malloc(sizeof(struct  c
_node));

        tail=tail->next;

        tail->data=*s++;

    }

    tail->next=NULL;

/*****      END      *****/

}

```

```

void  judgePalindrome(C_NODE  *head) {
/*****      BEGIN      *****/第二段代码/
char  s[100]={},  *pc=s;

    int  t=0;

    int  flag=1;

    struct  c_node  *p;

    for(p=head;p;p=p->next)

        *pc++=p->data;

    for(t=0;  t<strlen(s);  t++)

        if(s[t]!=s[strlen(s)-t-1]) {

            flag=0;

            break;

        }

    if(flag==0)

        printf("false");

    else

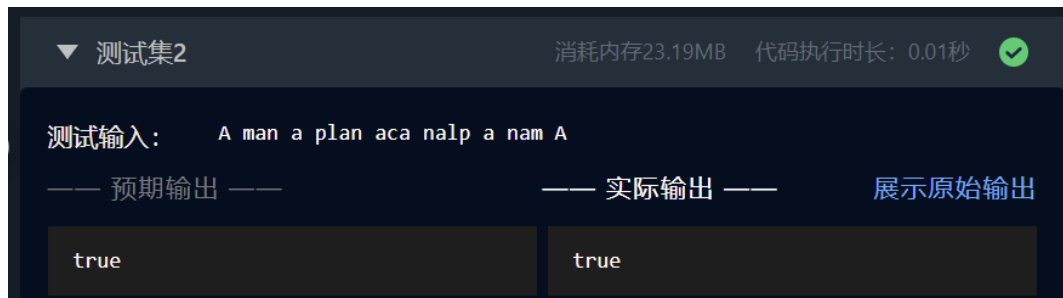
        printf("true");

/*****      END      *****/

}

```

运行结果如下：



5. 班级成绩单（成绩排序 2）

思路：由于此题需要在排序的基础上交换结点的指针域，这一操作涉及到所有链表中的元素，需要改变指针的连接方式。因此每一个子函数都需要重复这样的排序过程。代码如下：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct stuinfo{
    char id[20];
    char name[100];
    int score[4];
    int POINT_SUM;
    double POINT_AVG;
    struct stuinfo *next;
};

void *listup(struct stuinfo **head);
void START(struct stuinfo *head);
void END(struct stuinfo *head);
```

```

void change(struct stuinfo *head);
void giveAVG(struct stuinfo *head);
void giveSUM(struct stuinfo *head);

void main() {
    struct stuinfo *head;
    head=(struct stuinfo*)malloc(sizeof(struct stuinfo));
    head->next=0;
    int menu=0;

    while (scanf("%d",&menu)!=EOF)
    {
        switch(menu) {
            case 0:break;
            case 1:START(head); listup(&head); break;
            case 2:END(head); break;
            case 3:change(head); listup(&head); break;
            case 4:giveAVG(head); break;
            case 5:giveSUM(head); break;
        }
    }
}

void *listup(struct stuinfo **head) {
    int flag=0;
    struct stuinfo *pc, *pb, *pa;
    if ((*head)->next)
    for (pa=*head; pa->next->next; pa=pa->next) {
        pb=pa->next;
    }
}

```

```

        pc=pb->next;
        if(pb->POINT_AVG < pa->POINT_AVG) {
            flag+=1;
            pa->next=pb->next;
            pb->next=pa;
            *head=pb;
        }
        if(pc->POINT_AVG < pb->POINT_AVG) {
            flag+=1;
            pa->next=pc;
            pb->next=pc->next;
            pc->next=pb;
        }
    }
    if(flag!=0)
        listup(head);
}

void END(struct stuinfo *head) {
    struct stuinfo *p;
    for(p=head;p;p=p->next)
    {
        printf("%s %s ",p->id,p->name);
        printf("%d %d %d %d\n",p->score[0],p->score[1],p->
score[2],p->score[3]);
    }
}

void START(struct stuinfo *head) {

```



```

    struct stuinfo *p;
    static int t=0;
    int x,count;
    scanf("%d",&x);
    for(p=head; p->next; p=p->next);
    for(count=0; count<x; count++){
        if(t!=0){
            p->next=(struct stuinfo*)malloc(sizeof(struct stuinfo));
            p=p->next;
        }
        else
            t=1;
        scanf("%s",p->id);
        scanf("%s",p->name);
        scanf("%d %d %d %d",&p->score[0],&p->score[1],&p->score[2]
, &p->score[3]);
        p->POINT_SUM=p->score[0]+p->score[1]+p->score[2]+p->score[3];
        p->POINT_AVG=p->POINT_SUM/4.0;
        p->next=0;
    }
}

```

```

void giveSUM(struct stuinfo *head){
    struct stuinfo *p;
    for(p=head;p;p=p->next){
        printf("%s %s ",p->id,p->name);
        printf("%d ",p->POINT_SUM);
        printf("%.2f\n",p->POINT_AVG);
    }
}

```

```

}

void change(struct stuinfo *head) {
    struct stuinfo *p;
    int i;
    char temp[20];
    scanf("%s", temp);
    scanf("%d", &i);
    for(p=head; p; p=p->next) {
        if(!strcmp(temp, p->id)) {
            scanf("%d", &p->score[i-1]);
            p->POINT_SUM=p->score[0]+p->score[1]+p->score[2]+p->score[3];
            p->POINT_AVG=p->POINT_SUM/4.0;
        }
    }
}

void giveAVG(struct stuinfo *head) {
    struct stuinfo *p;
    for(p=head; p; p=p->next) {
        printf("%s %s ", p->id, p->name);
        printf("%.2f\n", p->POINT_AVG);
    }
}

```

运行结果如下：

▼ 测试集1

消耗内存22.95MB 代码执行时长: 0.01秒

测试输入: 1
2
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
2
3
U202054321 1 66
4
5
0

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75

U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75

▼ 测试集2

消耗内存22.95MB 代码执行时长: 0.01秒

测试输入: 1
4
U202054321 Rose 89 94 85 100
U202056789 Tom 12 34 56 78
U202012345 Jack 99 100 80 96
U202098765 Jerry 98 76 54 32
2
3
U202054321 1 66
4
5
0

—— 预期输出 ——

—— 实际输出 ——

展示原始输出

U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202056789 Tom 45.00
U202098765 Jerry 65.00
U202054321 Rose 86.25
U202012345 Jack 93.75
U202056789 Tom 180 45.00

U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202056789 Tom 45.00
U202098765 Jerry 65.00
U202054321 Rose 86.25
U202012345 Jack 93.75
U202056789 Tom 180 45.00

2.4 小结

结构、联合与链表是一种行之有效的组织数据的方式。其中结构可以很方便的把同类数据存在一起，而链表能相对快捷地访问其前后元素。同时，对于同一问题和同样的需求，也可能存在很多种不同的解决方法。因此，我需要灵活使用所学知识，具体问题具体分析。

参考文献

- [1] 卢萍, 李开, 王多强, 甘早斌. C 语言程序设计典型题解与实验指导, 北京: 清华大学出版社, 2019
- [2] 卢萍, 李开, 王多强, 甘早斌. C 语言程序设计, 北京: 清华大学出版社, 2021